

A Bicriteria Approximation for the Reordering Buffer Problem*

Siddharth Barman[†]Shuchi Chawla[‡]Seeun Umboh[§]

Abstract

In the reordering buffer problem (RBP), a server is asked to process a sequence of requests lying in a metric space. To process a request the server must move to the corresponding point in the metric. The requests can be processed slightly out of order; in particular, the server has a buffer of capacity k which can store up to k requests as it reads in the sequence. The goal is to reorder the requests in such a manner that the buffer constraint is satisfied and the total travel cost of the server is minimized. The RBP arises in many applications that require scheduling with a limited buffer capacity, such as scheduling a disk arm in storage systems, switching colors in paint shops of a car manufacturing plant, and rendering 3D images in computer graphics.

We study the offline version of RBP and develop bicriteria approximations. When the underlying metric is a tree, we obtain a solution of cost no more than 9OPT using a buffer of capacity $4k + 1$ where OPT is the cost of an optimal solution with buffer capacity k . Constant factor approximations were known previously only for the uniform metric (Avigdor-Elgrabli et al., 2012). Via randomized tree embeddings, this implies an $O(\log n)$ approximation to cost and $O(1)$ approximation to buffer size for general metrics. Previously the best known algorithm for arbitrary metrics by Englert et al. (2007) provided an $O(\log^2 k \log n)$ approximation without violating the buffer constraint.

1 Introduction

We consider the reordering buffer problem (RBP) where a server with buffer capacity k has to process a sequence of requests lying in a metric space. The server is initially stationed at a given vertex and at any point of time it can store at most k requests. In particular, if there are k requests in the buffer then the server must process one of them (that is, visit the corresponding vertex in the metric space) before reading in the next request from the input sequence. The objective is to process the requests in an order that minimizes the total distance travelled by the server.

RBP provides a unified model for studying scheduling with limited buffer capacity. Such scheduling problems arise in numerous areas including storage systems, computer graphics, job shops, and information retrieval (see [15, 18, 13, 5]). For example, in a secondary storage system the overall performance critically depends on the response time of the underlying disk devices. Hence disk devices need to schedule their disk arm in a way that minimizes the *mean seek time*. Specifically, these devices receive read/write requests which are located on different cylinders and they must move the disk arm to the proper cylinder in order to serve a request. The device can buffer a limited number of requests and must deploy a scheduling policy to minimize the overall service time. Note that we can model this disk arm scheduling problem as a RBP instance by representing the disk arm as a server and the array of cylinders as a metric space over read/write requests.

The RBP can be seen to be NP-Hard via a reduction from the traveling salesperson problem. We study approximation algorithms. RBP has been considered in both online and offline contexts. In the online setting the entire input sequence is not known beforehand and the requests arrive one after the other. This setting

*This work was supported in part by NSF awards CCF-0643763 and CNS-0905134.

[†]Computer Sciences Department, University of Wisconsin–Madison. sid@cs.wisc.edu

[‡]Computer Sciences Department, University of Wisconsin–Madison. shuchi@cs.wisc.edu

[§]Computer Sciences Department, University of Wisconsin–Madison. seeun@cs.wisc.edu

was considered by Englert et al. [7], who developed an $O(\log^2 k \log n)$ -competitive algorithm. To the best of our knowledge this is the best known approximation guarantee for RBP over arbitrary metrics both in the online and offline case.

RBP remains NP-Hard even when restricted to the uniform metric (see [6]). In fact the uniform metric is an interesting special case as it models scheduling of paint jobs in a car manufacturing plant. In particular, switching paint color is a costly operation; hence, paint shops temporarily store cars and process them out of order to minimize color switches. Over the uniform metric, RBP is somewhat related to paging. However, unlike for the latter, simple greedy strategies like First in First Out and Least Recently Used yield poor competitive ratios (see [15]). Even the offline version of the uniform metric case does not seem to admit simple approximation algorithms. The best known approximation for this setting, due to Avigdor-Elgrabli et al. [3], relies on intricate rounding of a linear programming relaxation in order to get a constant-factor approximation.

The hardness of the RBP appears to stem primarily from the strict buffer constraint; it is therefore natural to relax this constraint and consider bicriteria approximations. We say that an algorithm achieves an (α, β) bicriteria approximation if, given any RBP instance, it generates a solution of cost no more than αOPT using a buffer of capacity βk . Here OPT is the cost of an optimal solution with buffer capacity k . There are few bicriteria results known for the RBP. For the offline version of the uniform metric case, a bicriteria approximation of $(O(\frac{1}{\epsilon}), 2 + \epsilon)$ for every $\epsilon > 0$ was given by Chan et al. [6]. For the online version of this restricted case, Englert et al. [8] developed a $(4, 4)$ -competitive algorithm. They further showed how to convert this bicriteria approximation into a true approximation with a logarithmic ratio. We show in Appendix A that such a conversion from a bicriteria approximation to a true approximation is not possible at small loss in more general metrics, e.g. the evenly-spaced line metric. In more general metrics, relaxing the buffer constraint therefore gives us significant extra power in approximation.

We study bicriteria approximation for the offline version of RBP. When the underlying metric is a weighted tree we obtain a $(9, 4 + \frac{1}{k})$ bicriteria approximation algorithm. Using tree embeddings of [9] this implies a $(O(\log n), 4 + \frac{1}{k})$ bicriteria approximation for arbitrary metrics over n points.

Other Related Work: Besides the work of Englert et al. [7], existing results address RBP over very specific metrics. RBP was first considered by Räcke et al. [15]. They focused on the uniform metric with online arrival of requests and developed an $O(\log^2 k)$ -competitive algorithm. This was subsequently improved on by a number of results [8, 2, 1], leading to an $O(\sqrt{\log k})$ -competitive algorithm [1].

With the disk arm scheduling problem in mind, Khandekar et al. [12] considered the online version of RBP over the evenly-spaced line metric (line graph with unit edge lengths) and gave an online algorithm with a competitive ratio of $O(\log^2 n)$. This was improved on by Gamzu et al. [10] to an $O(\log n)$ -competitive algorithm.

Bicriteria approximations have been studied previously in the context of resource augmentation (see [14] and references therein). In this paradigm, the algorithm is augmented with extra resources (usually faster processors) and the benchmark is an optimal solution without augmentation. This approach has been applied to, for example, paging [17], scheduling [11, 4], and routing problems [16].

Techniques: We can assume without loss of generality that the server is lazy and services each request when it absolutely must—to create space in the buffer for a newly received request. Then after reading in the first k requests, the server must serve exactly one request for each new one received. Intuitively, adding extra space in the buffer lets us defer serving decisions. In particular, while the optimal server must serve a request at every step, we serve requests in batches at regular intervals. Partitioning requests into batches appears to be more tractable than determining the exact order in which requests appear in an optimal solution. This enables us to go beyond previous approaches (see [2, 3]) that try to extract the order in which requests appear in an optimal solution. We enforce the buffer capacity constraint by placing lower bounds on the cardinalities of the batches. In particular, by ensuring that each batch is large enough, we make sure that the server “carries forward” few requests. Then the maximum buffer utilization can be bounded by the number of requests carried forward plus the number read before the next batch is processed.

A crucial observation that underlies our algorithm is that when the underlying metric is a tree, we can find vertices $\{v_i\}_i$ that any solution with buffer capacity k must visit in order. This allows us to anchor the i th batch at v_i and equate the serving cost of a batch to the cost of the subtree spanning the batch and rooted at v_i . Overall, when the underlying metric is a tree, the problem of finding low-cost batches with cardinality constraints reduces to finding low-cost subtrees which are rooted at v_i s, cover all the requests, and satisfy the same cardinality constraints. We formulate a linear programming relaxation, LP1, for this covering problem.

Rounding LP1 directly is difficult because of the cardinality constraints. To handle this we round LP1 partially to formulate another relaxation that is free of the cardinality constraints and is amenable to rounding. Specifically, using a fractional optimal solution to LP1, we determine for each request j an *interval* of indices, $\Gamma(j)$, such that any solution that assigns every request to a batch within its corresponding interval approximately satisfies the buffer constraint. This allows us to remove the cardinality constraints and instead formulate an interval-assignment relaxation LP2. In order to get the desired bicriteria approximation we show two things: first, the optimal cost achieved by LP2 is within a constant factor of the optimal cost for the given RBP instance; second, an integral feasible solution of LP2 can be transformed into a RBP solution using a bounded amount of extra buffer space. Finally we develop a rounding algorithm for LP2 which achieves an approximation ratio of 2.

2 Notation

An instance of RBP is specified by a metric space over a vertex set V , a sequence of n vertices (requests), an integer k , and a starting vertex v_0 . The metric space is represented by a graph $G = (V, E)$ with distance function $d : E \rightarrow \mathbf{R}^+$ on edges. We index requests by j . We assume without loss of generality that requests are distinct vertices. Starting at v_0 , the server reads requests from the input sequence into its buffer and clears requests from its buffer by visiting them in the graph (we say these requests are served). The goal is to serve all requests, having at most k buffered requests at any point in time, with minimum traveling distance. We denote the optimal solution as OPT. For the most part of this paper, we focus on the special case where G is a tree.

We break up the timeline into windows as follows. Without loss of generality, n is a multiple of $2k + 1$, i.e. $n = (2k + 1)m$. For $i \in [m]$, we define window W_i to be the set of requests from $(2k + 1)(i - 1) + 1$ to $(2k + 1)i$. Let $w(j)$ be the index of the window in which j belongs. The i -th *time window* is defined to be the duration in which the server read W_i .

3 Reduction to Request Cover Problem

In this section we show how to use extra buffer space to convert the RBP into a new and simpler problem that we call Request Cover. The key tool for the reduction is the following lemma which states that we can find for every window a vertex in the graph G that must be visited by any feasible solution within the same window. We call these vertices *terminals*. This allows us to break up the server's path into segments that start and end at terminals.

Lemma 1. *For each i , there exists a vertex v_i such that all feasible solutions with buffer capacity k must visit v_i in the i -th time window.*

Proof. Fix a feasible solution and i . We orient the tree as follows. For each edge $e = (u, v)$, if after removing e from the tree, the component containing u contains at most k requests of W_i , then we direct the edge from u to v . Since $|W_i| = 2k + 1$, there is exactly one directed copy of each edge.

An oriented tree is acyclic so there exists a vertex v_i with incoming edges only. We claim that the server must visit v_i during the i -th time window. During the i -th time window, the server reads all $2k + 1$ requests of W_i . Since each component of the induced subgraph $G[V \setminus \{v_i\}]$ contains at most k requests of W_i and the server has a buffer of size k , it cannot remain in a single component for the entire time window. Therefore, the server must visit at least two components, passing by v_i , at some point during the i -th time window. \square

For the remainder of the argument, we will fix the terminals v_1, \dots, v_m . Note that since G is a tree, there is a unique path visiting the terminals in sequence, and every solution must contain this path. For each i , let P_i denote the path from v_{i-1} to v_i .

We can now formally define request covers.

Definition 1 (Request cover). *Let \mathcal{B} be a partition of the requests into batches B_1, \dots, B_m , and \mathcal{E} be an ordered collection of m edge subsets $E_1, \dots, E_m \subseteq E$. The pair $(\mathcal{B}, \mathcal{E})$ is a request cover if*

1. *For every request j , the index of the batch containing j is at least $w(j)$, i.e. the window in which j is released.*
2. *For all $i \in [m]$, $E_i \cup P_i$ is a connected subgraph spanning B_i .*
3. *There exists a constant β such that for all $i \in [m]$, we have $\sum_{l \leq i} |B_l| \geq (2k+1)i - \beta k$; we say that the request cover is β -feasible. We call the request cover feasible if $\beta = 1$.*

The length of a request cover is $d(\mathcal{E}) = \sum_i d(E_i)$.

Definition 2 (Request Cover Problem (RCP)). *In the RCP we are given a metric space $G = (V, E)$ with lengths $d(e)$ on edges, a sequence of n requests, buffer capacity constraint k , and a sequence of $m = n/(2k+1)$ terminals v_1, \dots, v_m . Our goal is to find a feasible request cover of minimum length.*

We will now relate the request cover problem to the RBP. Let $(\mathcal{B}^*, \mathcal{E}^*)$ denote the optimal solution to the RCP. We show on the one hand (Lemma 2) that this solution has cost within a constant factor of OPT, the optimal solution to RBP. On the other hand, we show (Lemma 3) that any β -feasible solution to RCP can be converted into a solution to the RBP that is feasible for a buffer of size $(2 + \beta)k + 1$ with a constant factor loss in length.

Lemma 2. $d(\text{OPT}) \geq d(\mathcal{E}^*)$.

Proof. For each i , let E_i be the edges traversed by the optimal server during the i -th time window and let \mathcal{E} be the collection of edge subsets. We have $d(\text{OPT}) \geq \sum_i d(E_i) = d(\mathcal{E})$, so it suffices to show that $\mathcal{E} = (E_1, \dots, E_m)$ is a feasible request cover. By Lemma 1, both E_i and P_i are connected subgraphs containing v_i for each i . Hence \mathcal{E} is connected. Since E_l contains the requests served in the l -th time window for each l , and for each i the server has read $(2k+1)i$ requests and served all except at most k of them by the end of the i -th time window, we get that $\sum_{l \leq i} |B_l| \geq (2k+1)i - k$. This proves that \mathcal{E} is a feasible request cover. \square

Next, consider a request cover $(\mathcal{B}, \mathcal{E})$. We may assume without loss of generality that for all i , $E_i \cap P_i = \emptyset$. This observation implies that E_i can be partitioned into components $E_i(p)$ for each vertex $p \in P_i$, where $E_i(p)$ is the component of E_i containing p .

We will now define a server for the RBP, BATCH-SERVER(\mathcal{B}, \mathcal{E}), based on the solution $(\mathcal{B}, \mathcal{E})$. Recall that the server has to start at v_0 . In the i -th iteration, it first buffers all requests in window W_i . Then it moves from v_{i-1} to v_i and serves requests of B_i as it passes by them.

Algorithm 1 BATCH-SERVER(\mathcal{B}, \mathcal{E})

- 1: Start at v_0
 - 2: **for** $i = 1$ **to** m **do**
 - 3: (Buffering phase) Read W_i into buffer
 - 4: (Serving phase) Move from v_{i-1} to v_i along P_i , and for each vertex $p \in P_i$, perform an Eulerian tour of $E_i(p)$. Serve requests of B_i along the way.
 - 5: **end for**
-

Lemma 3. *Given a β -feasible request cover $(\mathcal{B}, \mathcal{E})$, BATCH-SERVER(\mathcal{B}, \mathcal{E}) is a feasible solution to the RBP instance with a buffer of size $(2 + \beta)k + 1$, and has length at most $d(\text{OPT}) + 2d(\mathcal{E})$.*

Proof. We analyze the length first. In iteration i , the server uses each edge of P_i exactly once. Since E_i is a disjoint union of $E_i(p)$ for $p \in P_i$, the server uses each edge of E_i twice during the Eulerian tours of E_i 's components. The total length is therefore

$$\sum_i d(P_i) + \sum_i 2d(E_i) \leq d(\text{OPT}) + 2d(\mathcal{E}).$$

Next, we show that the server has at most $(2 + \beta)k + 1$ requests in its buffer at any point in time. We claim that all of B_i is served by the end of the i -th iteration. Consider a request j that belongs to a batch B_i . Since i is at least as large as $w(j)$, the request has already been received by the i th phase. The server visits j 's location during the i th iteration and therefore services the request at that time if not earlier. This proves the claim.

The claim implies that the server begins the $(i + 1)$ -th iteration having read $(2k + 1)i$ requests and served $\sum_{l \leq i} |B_l| \geq (2k + 1)i - \beta k$ requests, that is, with at most βk requests in its buffer. It adds $2k + 1$ requests to be the buffer during this iteration. So it uses at most $(2 + \beta)k + 1$ buffer space at all times. \square

4 Approximating the Request Cover Problem

We will now show how to approximate the request cover problem. Our approach is to start with an LP relaxation of the problem, and use the optimal fractional solution to the LP to further define a simpler covering problem which we then approximate in Section 4.2.

4.1 The request cover LP and the interval cover problem

The integer linear program formulation of RCP is as follows. To obtain an LP relaxation we relax the last two constraints to $x(i, j), y(e, i) \in [0, 1]$.

$$\begin{array}{ll} \text{minimize} & \sum_i \sum_e y(e, i) d_e \\ \text{subject to} & \sum_{w(j) \leq i} x(j, i) \geq 1 \quad \forall j \\ & \sum_{j: w(j) \leq i} \sum_{i' \leq i} x(j, i') \geq (2k + 1)i - k \quad \forall i \\ & y(e, i) \geq x(j, i) \quad \forall i, j, e \in R_{ji} \\ & x(j, i) \in \{0, 1\} \quad \forall i, j \\ & y(e, i) \in \{0, 1\} \quad \forall i, e \end{array} \quad (\text{LP1})$$

Here the variable $x(j, i)$ indicates whether request j is assigned to batch B_i and the variable $y(e, i)$ indicates whether edge e is in E_i . Recall that the edge set E_i along with path P_i should span B_i . Let R_{ji} denote the (unique) path in G from j to P_i . The third inequality above captures the constraint that if j is assigned to B_i and $e \in R_{ji}$, then e must belong to E_i .

Let (x^*, y^*) be the fractional optimal solution to the linear relaxation of (LP1). Instead of rounding (x^*, y^*) directly to get a feasible request cover, we will show that it is sufficient to find request covers that “mimic” the fractional assignment x^* but do not necessarily satisfy the cardinality constraints on the batches (i.e. the second set of inequalities in the LP). To this end we define an *interval request cover* below.

Definition 3 (Interval request cover). *For each request j , we define the service deadline $h(j) = \min\{i \geq w(j) : \sum_{l \leq i} x^*(j, l) \geq 1/2\}$ and the service interval $\Gamma(j) = [w(j), h(j)]$. A request cover $(\mathcal{B}, \mathcal{E})$ is an interval request cover if it assigns every request to a batch within its service intervals.*

In other words, while x^* “half-assigns” each request no later than its service deadline, an interval request cover mimics x^* by integrally assigning each request no later than its service deadline. The following is a linear programming formulation for the problem of finding minimum length interval request covers.

$$\begin{array}{ll}
\text{minimize} & \sum_i \sum_e y(e, i) d_e \\
\text{subject to} & \sum_{i \in \Gamma(j)} x(j, i) \geq 1 \quad \forall j \\
& y(e, i) \geq x(j, i) \quad \forall j, i \in \Gamma(j), e \in R_{ji} \\
& x(j, i), y(e, i) \in [0, 1] \quad \forall i, j, e
\end{array} \tag{LP2}$$

Let (\tilde{x}, \tilde{y}) be the fractional optimal of (LP2). We now show that interval request covers are 2-feasible request covers and that $d(\tilde{y}) \leq 2d(y^*)$. Since $d(y^*) \leq d(\mathcal{E}^*)$, it would then suffice to round (LP2).

Lemma 4. *Interval request covers are 2-feasible.*

Proof. Fix i . Let $H_i := \{j : h(j) \leq i\}$ denote the set of all requests whose service intervals end at or before the i th time window. We first claim that $|H_i| \geq (2k+1)i - 2k$. In particular, the second constraint of (LP1) and the definition of H_i gives us

$$\begin{aligned}
(2k+1)i - k &= \sum_{j:w(j) \leq i} \sum_{i' \leq i} x^*(j, i') = \sum_{j \in H_i: w(j) \leq i} \sum_{i' \leq i} x^*(j, i') + \sum_{j \notin H_i: w(j) \leq i} \sum_{i' \leq i} x^*(j, i') \\
&\leq \sum_{j \in H_i: w(j) \leq i} 1 + \sum_{j \notin H_i: w(j) \leq i} \frac{1}{2} = |H_i| + \frac{1}{2} ((2k+1)i - |H_i|).
\end{aligned}$$

The claim now follows from rearranging the above inequality.

Note that in an interval request cover, each request in H_i is assigned to some batch B_l with $l \leq i$. Therefore,

$$\sum_{l \leq i} |B_l| \geq |H_i| \geq (2k+1)i - 2k.$$

□

We observe that multiplying all the coordinates of x^* and y^* by 2 gives us a feasible solution to (LP2). Thus we have the following lemma.

Lemma 5. *We have $d(\tilde{y}) \leq 2d(y^*)$.*

Note that the lemma says nothing about the integral optimal of (LP2) so a solution that merely approximates the optimal integral interval request cover may not give a good approximation to the RBP, and we need to bound the integrality gap of the LP. In the following subsection, we show that we can find an interval request cover of length at most $2d(\tilde{y})$.

4.2 Approximating the Interval Assignment LP

Before we describe the general approximation, we consider two special cases for insight.

Example: single edge. Suppose the tree consists of a single unit-length edge $e = (u, v)$, all requests reside at u , and all terminals at v . In this case, $R_{ji} = \{e\}$ for all pairs j and i so the second set of constraints in (LP2) is simply

$$y(i) \geq x(j, i) \quad \forall j, i \in \Gamma(j)$$

where we write $y(i)$ for $y(e, i)$. A minimum solution satisfies these constraints with equality. Summing over $i \in \Gamma(j)$, we get that in this case (LP2) is equivalent to

$$\begin{array}{ll} \text{minimize} & \sum_i y(i) \\ \text{subject to} & \sum_{i \in \Gamma(j)} y(i) \geq 1 \quad \forall j \end{array}$$

This is exactly the linear relaxation for the hitting set¹ problem where the sets we want to hit are intervals. While the general hitting set problem is hard, it turns out that this special case can be solved exactly in polynomial time and the relaxation has no integrality gap². Thus, we get an optimal solution via a reduction to the minimum interval hitting set problem: compute a minimum hitting set M for the set of intervals $\mathcal{I} := \{\Gamma(j)\}$, and then add e to E_i for $i \in M$.

Example: two edges. Suppose the tree is a line graph consisting of three vertices u_1, u_2 and v with unit-length edges $e_1 = (u_1, v)$ and $e_2 = (u_2, u_1)$ (Figure 1(a)). Requests reside at u_1 and u_2 , and all terminals at v . For each i and j residing at u_1 , we have $R_{ji} = \{e_1\}$. For each i and j residing at u_2 we have $R_{ji} = \{e_1, e_2\}$. Thus feasible solutions to (LP2) satisfy the constraints

$$\begin{aligned} \sum_{i \in \Gamma(j)} y(e_1, i) &\geq 1 \quad \forall j, \\ \sum_{i \in \Gamma(j)} y(e_2, i) &\geq 1 \quad \forall j \in u_2. \end{aligned}$$

The constraints suggest that the vector $y(e_1, \cdot)$ is a fractional hitting set for the collection of intervals $\mathcal{I}(e_1) := \{\Gamma(j)\}$, and $y(e_2, \cdot)$ for $\mathcal{I}(e_2) := \{\Gamma(j) : j \in u_2\}$. In light of the single-edge special case, a naive approach is to first compute minimum hitting sets $M(e_1)$ and $M(e_2)$ for $\mathcal{I}(e_1)$ and $\mathcal{I}(e_2)$, respectively. Then we add e_1 to E_i for $i \in M(e_1)$, and e_2 to E_i for $i \in M(e_2)$. However, the resulting edge sets may not be connected. Instead, we make use of the following crucial facts:

- (1) We should include e_2 in E_i only if $e_1 \in E_i$, and,
- (2) Minimal hitting sets are at most twice minimum fractional hitting sets (see Lemma 9).

These facts suggest that we should first compute a minimal hitting set $M(e_1)$ for $\mathcal{I}(e_1)$ and then compute a minimal hitting set $M(e_2)$ for $\mathcal{I}(e_2)$ with the constraint that $M(e_2) \subseteq M(e_1)$. This is a valid solution to (LP2) since $\mathcal{I}(e_2) \subseteq \mathcal{I}(e_1)$. We proceed as usual to compute \mathcal{E} . The resulting \mathcal{E} is connected by (1) and $d(\mathcal{E}) \leq 2d(\tilde{y})$ by (2).

General case. Motivated by the two-edge example, at a high level, our approach for the general case is as follows:

1. We construct interval hitting set instances over each edge.
2. We solve these instances starting from the edges nearest to the paths P_i first.
3. We iteratively “extend” solutions for the instances nearer the paths to get minimal hitting sets for the instances further from the paths.

¹A subset X of a universe U is a *hitting set* for $\mathcal{S} \subset 2^U$ if $X \cap S \neq \emptyset$ for all $S \in \mathcal{S}$.

²One way to see this is that the columns of the constraint matrix has consecutive ones, and thus the constraint matrix is totally unimodular.

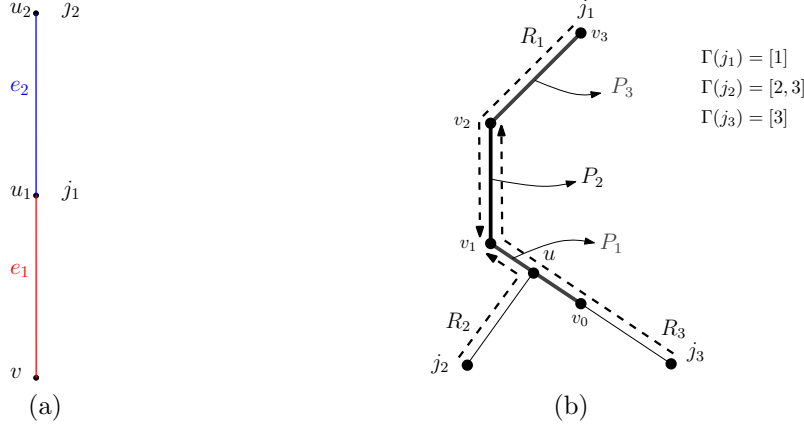


Figure 1: (a) The two-edge example; (b) R_i is the path from j_i to P_i , for $i \in \{1, 2, 3\}$. Note that arc (v_1, v_2) precedes (u, v_1) and no arc precedes (v_1, v_2) .

We then use Lemma 9 to argue a 2-approximation on an edge-by-edge basis.

Figure 1(b) gives an example of an instance of interval request cover. Note that whether an edge is closer to some P_i along a path R_{ji} for some j depends on which direction we are considering the edge in. We therefore modify (LP2) to include directionality of edges, replacing each edge e with bidirected arcs and directing the paths R_{ji} from j to P_i .

$$\begin{array}{ll}
 \text{minimize} & \sum_i \sum_a y(a, i) d_a \\
 \text{subject to} & \sum_{i \in \Gamma(j)} x(j, i) \geq 1 \quad \forall j \\
 & y(a, i) \geq x(j, i) \quad \forall j, i \in \Gamma(j), a \in R_{ji} \\
 & x(j, i), y(a, i) \in [0, 1] \quad \forall i, j, a
 \end{array} \tag{LP2'}$$

For every edge e and window i , there is a single orientation of edge e that belongs to R_{ji} for some j . So there is a 1-1 correspondence between the variables $y(e, i)$ in (LP2) and the variables $y(a, i)$ in (LP2'), and the two LPs are equivalent. Henceforth we focus on (LP2').

Before presenting our final approximation we need some more notation.

Definition 4. For each request j , we define R_j to be the directed path from j to $\bigcup_{i \in \Gamma(j)} P_i$. For each arc a , we define $C(a) = \{j : a \in R_j\}$ and the set of intervals $\mathcal{I}(a) = \{\Gamma(j) : j \in C(a)\}$. We say that a is a cut arc if $C(a) \neq \emptyset$.

We say that an arc a precedes arc a' , written $a \prec a'$, if there exists a directed path in the tree containing both the arcs and a appears after a' in the path.

Lemma 6. Feasible solutions (x, y) of (LP2') satisfy the following set of constraints for all arcs a :

$$\sum_{i \in \Gamma(j)} y(a, i) \geq 1 \quad \forall j \in C(a).$$

Proof. Let (x, y) be a feasible solution of (LP2'). Fix an arc a and $j \in C(a)$. For each $i \in \Gamma(j)$, we have $a \in R_{ji}$ since R_j is a path from j to a connected subgraph containing P_i . By feasibility, we have $y(a, i) \geq x(j, i)$. Summing over $\Gamma(j)$, we get $\sum_{i \in \Gamma(j)} y(a, i) \geq \sum_{i \in \Gamma(j)} x(j, i) \geq 1$ where the last inequality follows from feasibility. \square

We are now ready to describe the algorithm. At a high level, Algorithm 2 does the following: initially, it finds a cut arc a with no cut arc preceding it and computes a minimal hitting set $M(a)$ for $\mathcal{I}(a)$; iteratively, it finds a cut arc a whose preceding cut arcs have been processed previously, and minimally “extends” the hitting sets $M(a')$ computed previously for the preceding arcs a' to form a minimal hitting set $M(a)$.

Algorithm 2 Greedy extension

```

1:  $U \leftarrow \{a : C(a) \neq \emptyset\}$ 
2:  $A_i \leftarrow \emptyset$  for all  $i$ 
3:  $M(a) \leftarrow \emptyset$  for all arcs  $a$ 
4: while  $U \neq \emptyset$  do
5:   Let  $a$  be any arc in  $U$ 
6:   while there exists  $a' \prec a$  in  $U$  do
7:      $a \leftarrow a'$ 
8:   end while
9:   Let  $a = (u, v)$ 
10:   $F(a) \leftarrow \{i : v \in P_i\} \cup \bigcup_{w: (v,w) \prec a} M((v, w))$ 
11:  Set  $M(a) \subseteq F(a)$  to be a minimal hitting set for the intervals  $\mathcal{I}(a)$ 
12:   $A_i \leftarrow A_i \cup \{a\}$  for all  $i \in M(a)$ 
13:   $U \leftarrow U \setminus \{a\}$ 
14: end while
15:  $f(j) \leftarrow \min\{i \in \Gamma(j) : j \text{ incident to } A_i \text{ or } P_i\}$  for all  $j$ 
16:  $B_i \leftarrow \{j : f(j) = i\}$  for all  $i$ 
17: return  $\mathcal{A} = (A_1, \dots, A_m), \mathcal{B} = (B_1, \dots, B_m)$ 

```

We prove that Algorithm 2 actually manages to process all cut arcs a and that $F(a)$ is a hitting set for $\mathcal{I}(a)$. First, we make the following observation.

Lemma 7. *For each iteration, the following holds.*

1. *If $U \neq \emptyset$, the inner ‘while’ loop finds an arc.*
2. *$F(a)$ is a hitting set for the intervals $\mathcal{I}(a)$.*

Proof. Since we have a bidirected tree and an arc does not precede its reverse arc, the inner ‘while’ loop does not repeat arcs and hence it stops with some arc. This proves the first statement.

We prove the second statement by induction on the algorithm’s iterations. In the first iteration, the set U consists of cut arcs so $a' \not\prec a$ for all cut arcs a' . Therefore, for all $\Gamma(j) \in \mathcal{I}(a)$, a is the arc on R_j closest to $\bigcup_{i \in \Gamma(j)} P_i$ and $v \in \bigcup_{i \in \Gamma(j)} P_i$. This proves the base case. Now we prove the inductive case. Fix an interval $\Gamma(j) \in \mathcal{I}(a)$. If a is the arc on R_j closest to $\bigcup_{i \in \Gamma(j)} P_i$ and $v \in \bigcup_{i \in \Gamma(j)} P_i$, then $F(a) \cap \Gamma(j) \neq \emptyset$. If not, then there exists a neighboring arc $(v, w) \in R_j$ closer to $\bigcup_{i \in \Gamma(j)} P_i$. We have that $\Gamma(j) \in \mathcal{I}((v, w))$ and $(v, w) \prec a$. Since the algorithm has processed all cut arcs preceding a , by the inductive hypothesis we have $F((v, w)) \cap \Gamma(j) \neq \emptyset$. This implies that $M((v, w))$ is a hitting set for $\mathcal{I}((v, w))$ and so $F(a) \cap \Gamma(j) \neq \emptyset$. Hence, $F(a)$ is a hitting set for $\mathcal{I}(a)$. \square

Let E_i be the set of edges whose corresponding arcs are in A_i and $\mathcal{E} = (E_1, \dots, E_m)$, i.e. the undirected version of \mathcal{A} .

Lemma 8. *$(\mathcal{B}, \mathcal{E})$ is an interval request cover.*

Proof. The connectivity of $E_i \cup P_i$ follows from the fact that the algorithm starts with $A_i = \emptyset$, and in each iteration an arc $a = (u, v)$ is added to A_i only if $v \in P_i$ or v is incident to some edge previously added to A_i .

Now it remains to show that $f(j) \in \Gamma(j)$ for all requests j , i.e. that there exists $i \in \Gamma(j)$ such that j is incident to A_i or P_i . If $R_j = \emptyset$, then $j \in \bigcup_{i \in \Gamma(j)} P_i$. On the other hand if $R_j \neq \emptyset$, then let $a \in R_j$ be the

arc incident to j . Since the algorithm processes all cut arcs, we have $a \in \bigcup_{i \in \Gamma(j)} A_i$ and thus j is incident to $\bigcup_{i \in \Gamma(j)} A_i$. In both cases, we have $f(j) \in \Gamma(j)$. \square

Next, we analyze the cost of the algorithm. Let $D(a)$ be the number of disjoint intervals in $\mathcal{I}(a)$.

Lemma 9. $D(a) \geq |M(a)|/2$ for all arcs a .

Proof. Let $i_1 < \dots < i_{|M(a)|}$ be the elements of $M(a)$. For each $1 \leq l \leq |M(a)|$, there exists an interval $\Gamma(j_l) \in \mathcal{I}(a)$ such that $M(a) \cap \Gamma(j_l) = \{i_l\}$, because otherwise $M(a) \setminus \{i_l\}$ would still be a hitting set, contradicting the minimality of $M(a)$. We observe that the intervals $\Gamma(j_l)$ and $\Gamma(j_{l+2})$ are disjoint since $\Gamma(j_l)$ contains i_l and $\Gamma(j_{l+2})$ contains i_{l+2} but neither contains i_{l+1} . Therefore, the set of $\lceil |M(a)|/2 \rceil$ intervals $\{\Gamma(j_l) : 1 \leq l \leq |M(a)| \text{ and } l \text{ odd}\}$ is disjoint. \square

Lemma 10. $d(\mathcal{E}) \leq 2d(\tilde{y})$.

Proof. Fix an arc a . From Lemmas 6 and 9, we get

$$\sum_i \tilde{y}(a, i) \geq D(a) \geq |M(a)|/2.$$

Since $d(\mathcal{E}) = d(\mathcal{A})$, we have

$$\begin{aligned} d(\mathcal{E}) &= \sum_a |M(a)| \cdot d_a \\ &\leq \sum_a \left(2 \sum_i \tilde{y}(a, i) \right) \cdot d_a = 2d(\tilde{y}). \end{aligned}$$

\square

Together with Lemmas 4 and 5, we have that $(\mathcal{B}, \mathcal{E})$ is a 2-strict request cover of length at most $4d(y^*) \leq 4d(\mathcal{E}^*)$. Lemmas 2 and 3 imply that BATCH-SERVER(\mathcal{B}, \mathcal{E}) travels at most 9 OPT and uses a buffer of capacity $4k + 1$. This gives us the following theorem.

Theorem 11. *There exists an offline $(9, 4 + \frac{1}{k})$ -bicriteria approximation for RBP when the underlying metric is a weighted tree.*

Using tree embeddings of [9], we get

Theorem 12. *There exists an offline $(O(\log n), 4 + \frac{1}{k})$ -bicriteria approximation for RBP over general metrics.*

References

- [1] A. Adamaszek, A. Czumaj, M. Englert, and H. Räcke. Almost tight bounds for reordering buffer management. In *STOC 2011*.
- [2] N. Avigdor-Elgrabli and Y. Rabani. An improved competitive algorithm for reordering buffer management. In *SODA 2010*.
- [3] N. Avigdor-Elgrabli and Y. Rabani. A constant factor approximation algorithm for reordering buffer management. *CoRR*, abs/1202.4504, 2012.
- [4] N. Bansal and K. Pruhs. Server scheduling in the l_p norm: a rising tide lifts all boat. In *STOC 2003*.
- [5] D. Blandford and G. Blelloch. Index compression through document reordering. In *Proceedings of the Data Compression Conference, DCC 2002*.

- [6] H.L. Chan, N. Megow, R. van Stee, and R. Sitters. The sorting buffer problem is np-hard. *CoRR*, abs/1009.4355, 2010.
- [7] M. Englert, H. Räcke, and M. Westermann. Reordering buffers for general metric spaces. In *STOC 2007*.
- [8] M. Englert and M. Westermann. Reordering buffer management for non-uniform cost models. *ICALP 2005*.
- [9] J. Fakcharoenphol, S. Rao, and K. Talwar. A tight bound on approximating arbitrary metrics by tree metrics. *J. Comput. Syst. Sci.*, 69(3), 2004.
- [10] I. Gamzu and D. Segev. Improved online algorithms for the sorting buffer problem. *STACS 2007*.
- [11] B. Kalyanasundaram and K. Pruhs. Speed is as powerful as clairvoyance. *Journal of the ACM*, 47(4), 2000.
- [12] R. Khandekar and V. Pandit. Online and offline algorithms for the sorting buffers problem on the line metric. *Journal of Discrete Algorithms*, 8(1), 2010.
- [13] J. Krokowski, H. Räcke, C. Sohler, and M. Westermann. Reducing state changes with a pipeline buffer. In *VMV 2004*.
- [14] K. Pruhs, J. Sgall, and E. Torng. Handbook of scheduling: Algorithms, models, and performance analysis. 2004.
- [15] H. Räcke, C. Sohler, and M. Westermann. Online scheduling for sorting buffers. In *ESA 2002*.
- [16] T. Roughgarden and É. Tardos. How bad is selfish routing? *Journal of the ACM*, 49(2), 2002.
- [17] D.D. Sleator and R.E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2), 1985.
- [18] S. Spieckermann, K. Gutenschwager, and S. Vosz. A sequential ordering problem in automotive paint shops. *International journal of production research*, 42(9), 2004.

A Gap Between Bicriteria and True Approximations

In this section we prove that there exists an instance on the evenly-spaced line metric in which the optimal offline solution with a buffer of size $k/4$ has to travel $\Omega(k)$ times the distance of the optimal offline solution with a buffer of size k .

We consider a line graph L with 2^k vertices $p_1 < \dots < p_{2^k}$ and unit-length edges. The input is a sequence of requests described by a binary tree S of depth k . Let r be the root of S . We denote the subtree rooted at a vertex v by $S(v)$. Let l_i be the i -th leaf according to the preordering of the tree. We define the *destination label* of vertex v to be $t(v) = \max\{i : l_i \in S(v)\}$ and the *origin label* of v to be $s(v) = \min\{i : l_i \in S(v)\}$. That is, $t(v)$ and $s(v)$ are the highest and lowest indices of any leaf in the subtree rooted at v , respectively. The input sequence is constructed as follows. First we obtain the sequence of vertices according to the preordering of the tree. Then we replace each non-leaf vertex v in the sequence with a request lying at $p_{t(v)}$ on the line, and each leaf vertex l_i with a block (which we refer to as a *leaf block*) of k requests lying at p_i on the line. For non-leaf vertices, we overload notation and use v to refer both to the vertex in the binary tree and the corresponding request.

Let $\text{OPT}(k)$ and $\text{OPT}(k/4)$ be the optimal offline solutions to the above input sequence that use buffers of capacity k and $k/4$, respectively.

Theorem 13. *We have $\text{OPT}(k/4) \geq \Omega(k) \text{OPT}(k)$.*

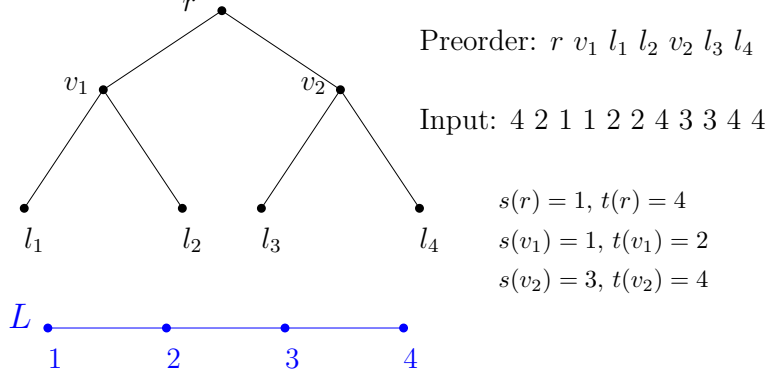


Figure 2: Example for $k = 2$

Example 1. For $k = 2$, the line metric is represented by the integers 1, 2, 3, 4 and the input sequence is 4, 2, 1, 1, 2, 2, 4, 3, 3, 4, 4.

We present a server k -SERVER that uses a buffer of size at most k and travels a distance of $2^k - 1$ on the above input sequence.

Algorithm 3 k -SERVER

- 1: **for** $i = 1$ **to** 2^k **do**
 - 2: Move to p_i
 - 3: Serve all requests v from the input that has $t(v) = i$
 - 4: **end for**
-

Lemma 14. On the above input sequence, k -SERVER uses a buffer of size at most k and travels a distance of $2^k - 1$.

Proof. Since k -SERVER visits each vertex of the line graph exactly once, it travels a distance of $2^k - 1$.

At the beginning of the i -th iteration, k -SERVER has just finished reading the $(i - 1)$ -th leaf block and is at p_i . Furthermore, it has also served all requests that reside at p_1, \dots, p_{i-1} . Thus, it needs to maintain in its buffer only the requests v up till the i -th leaf block that have $t(v) > i$. Since the input sequence is constructed using the preordering of S , these requests correspond to the ancestors of leaf l_i in S . The tree S is of depth k so it needs to maintain at most $k - 1$ requests in its buffer at all times, in addition to a space of 1 that is needed to read requests from the input. \square

Next, we show that $\text{OPT}(k/4) \geq \frac{k}{4} \text{OPT}(k)$. Let SERVER be an optimal server with a buffer of size $k/4$ for the above input sequence. We analyze the movement of SERVER in phases. We define the i -th phase to be the duration starting from the time the last request of the $(i - 1)$ th leaf block is read to the time the last request of the i -th leaf block is read.

Lemma 15. At the end of the i -th phase, SERVER is at p_i .

Proof. Since the requests of the i -th leaf block all lie at p_i on the line metric, we assume w.l.o.g. that either the entire block is served together or buffered together. However, the block is of length k , thus SERVER must serve the entire block. \square

For request v , let $d(v)$ be the distance travelled between $p_{t(v)}$ and the previously served request, and $D(v) = \sum_{u \in S(v)} d(v)$. Then, the total cost to serve non-leaf vertices is $D(r) = \sum_v d(v)$. We define C_i to be

the contents of SERVER's buffer at the end of the i -th phase, i.e. when it reads the last request of the i -th block. Let $C_i(v) = C_i \cap S(v)$ and $C(v) = \sum_{i \in [s(v), t(v)]} |C_i(v)|$.

Let $h(v)$ denote the height of v .

Lemma 16. *We have $C(v) + D(v) \geq \frac{h(v)}{2} 2^{h(v)}$ for all vertices v .*

Proof. We use a proof by induction on the height of v . For the base case, v is a leaf. The base case follows from the fact that a leaf has height 0 and both $C(v)$ and $D(v)$ are non-negative. We consider the inductive case next. Let v_1 and v_2 be the left and right children of v , respectively. Request v is read in the $s(v)$ -th phase. Suppose v is served in the i' -th phase. Since $C_i(v) = C_i(v_1) \cup C_i(v_2) \cup \{v\}$ if $v \in C_i(v)$ and $C_i(v) = C_i(v_1) \cup C_i(v_2)$ if $v \notin C_i(v)$, we get that

$$C(v) = \sum_{i \in [s(v), t(v)]} |C_i(v_1)| + |C_i(v_2)| + |[s(v), i' - 1]|.$$

We observe that $[s(v), t(v)] = [s(v_1), t(v_1)] \cup [s(v_2), t(v_2)]$, $s(v_1) = s(v)$ and $t(v_2) = t(v)$. Suppose that $i' \in [s(v_1), t(v_1)]$ and the request served just before v is v' . Lemma 15 implies that the server is at $p_{i'-1}$ at the beginning of the i' -th phase, so w.l.o.g. $t(v') \leq t(v)$. The input sequence is obtained using the preordering of S , so the server has not read any request u with $t(u) \in [s(v_2), t(v_2)]$. Hence, we have $t(v') < s(v_2)$ so the server must have traversed at least $p_{s(v_2)}, p_{s(v)+1}, \dots, p_{t(v_2)}$ to serve v . So, we have that $d(v) \geq 2^{h(v)-1}$.

On the other hand, if $i' \in [s(v_2), t(v_2)]$ then, $|[s(v), i' - 1]| \geq |[s(v_1), t(v_1)]| = 2^{h(v)-1}$. Thus, either $|[s(v), i' - 1]|$ or $d(v)$ is at least $2^{h(v)-1}$. Since $D(v) = D(v_1) + D(v_2) + d(v)$, we get

$$\begin{aligned} C(v) + D(v) &= C(v_1) + C(v_2) + |[s(v), i' - 1]| + D(v_1) + D(v_2) + d(v) \\ &\geq C(v_1) + C(v_2) + D(v_1) + D(v_2) + 2^{h(v)-1} \\ &\geq \frac{(h(v) - 1)}{2} 2^{h(v)} + 2^{h(v)-1} \\ &= \frac{h}{2} 2^{h(v)}, \end{aligned}$$

where the second inequality follows from applying the inductive hypothesis on both v_1 and v_2 . \square

SERVER cannot buffer more than $k/4$ requests at any point in time therefore $|C_i| \leq k/4$ for all i . Applying Lemma 16 to the root r implies that $D(r) \geq \frac{k}{2} 2^k - \frac{k}{4} 2^k = \frac{k}{4} 2^k$. Since $\text{OPT}(k/4) \geq D(r)$, this completes the proof of Theorem 13.